Automated Program Verification Environment
APROVE

Presented by:

Mostafa Elsaie

10th of May, 2011

Outline

- Termination Provers
  - Aim?
- APROVE
  - Input Languages
  - Syntax
  - Proving Mechanism
  - GUI
  - Reading the Output

Termination Provers

- The aim is to find the answer for “Does program terminate?“
- Based on Termination *Proof*

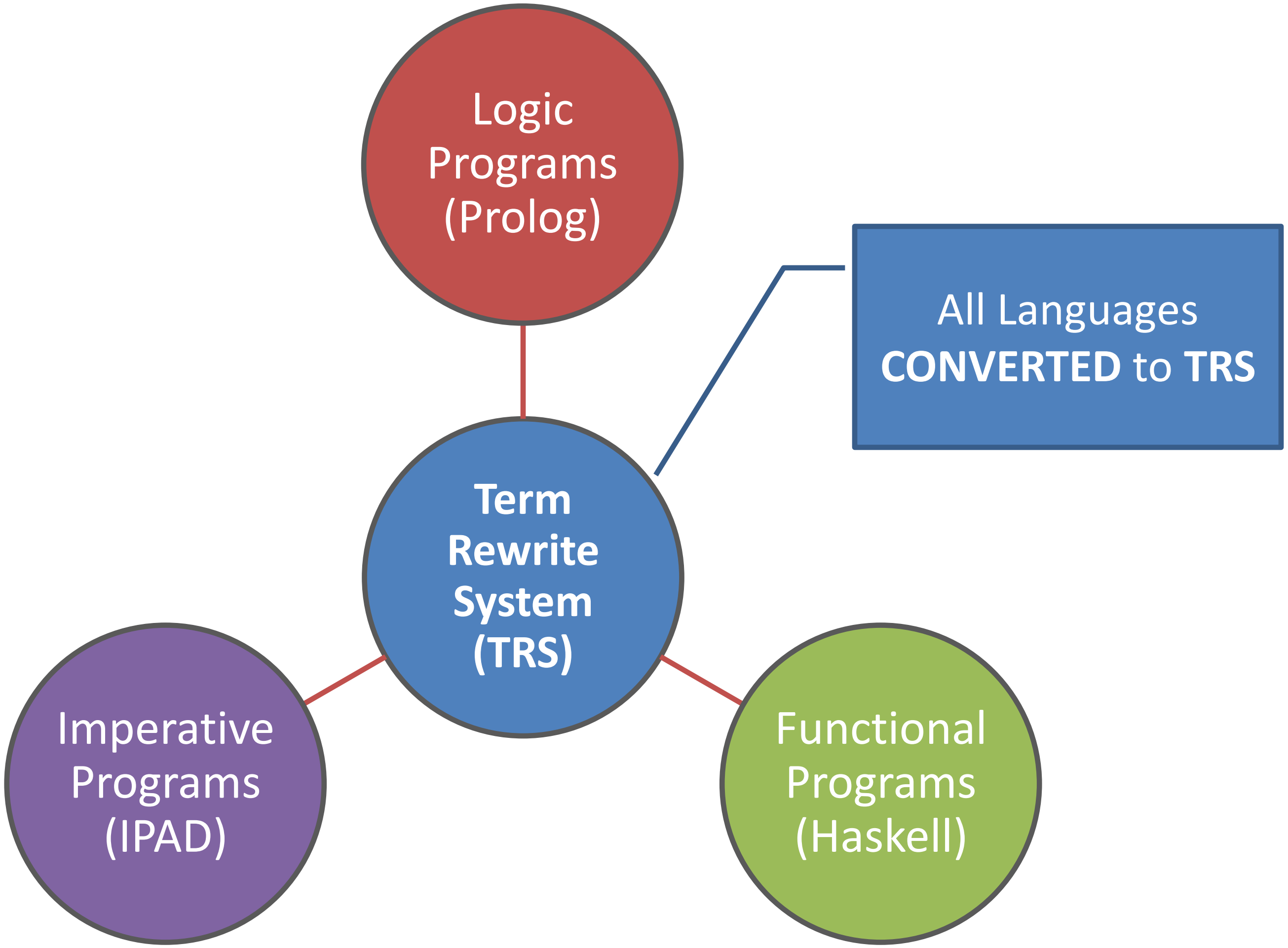
- *Example:*

```
i := 0
loop until i = SIZE_OF_DATA
   process_data(data[i])) //process the data chunk at position i
   i := i + 1 //move to the next chunk of data to be processed
```

APROVE - History

- Built at the *“RWTH Aachen University, Germany”*
- Developed by *“Research Group II of CS”*
- First Version released in 2001
- Most powerful prover for 04’,05’,06’…2010

APROVE - Input Languages



APROVE - TRS

- A TRS is a pair *(∑ , R)*

- The alphabet ∑ consists of:
  - Infinite set of variables *(x, y, z…)*
  - Non-empty set of Function symbols ***(F, G…)***

- The set of terms *Ter(∑)* is defined:
  - *(x, y, z..)* ∈ *Ter (∑)*
  - $F(t_1, t_2, t_3 \ldots) \in Ter(\Sigma)$

APROVE - TRS

- A substitution α is:
  - Map from *Ter* $(\sum)$ to *Ter* $(\sum)$
  - $\alpha(\mathbf{F}(t_1, t_2, t_3\ldots)) == \mathbf{F}(\alpha(t_1), \alpha(t_2), \alpha(t_3)\ldots)$

- A rewrite rule (A, B):
  - $A \rightarrow B$
  - LHS is not a variable
  - Variables in RHS already contained in LHS

APROVE - TRS

- Example
  - Let $\sum = \{\boldsymbol{A}, \boldsymbol{M}, \boldsymbol{S}, 0\}$

- *r1*: $\boldsymbol{A}(x,0) \rightarrow x$
- *r2*: $\boldsymbol{A}(x, \boldsymbol{S}(y)) \rightarrow \boldsymbol{S}(\boldsymbol{A}(x, y))$
- *r3*: $\boldsymbol{M}(x, 0) \rightarrow 0$
- *r4*: $\boldsymbol{M}(x, \boldsymbol{S}(y)) \rightarrow \boldsymbol{A}(\boldsymbol{M}(x, y), x)$
- *R5*: $\boldsymbol{S}(x) \rightarrow \boldsymbol{A}(x,1)$

- *Reduce* $\boldsymbol{M}\left(\boldsymbol{S}(\boldsymbol{S}(0)), \boldsymbol{S}(\boldsymbol{S}(0))\right) \rightarrow \boldsymbol{S}(\boldsymbol{S}(\boldsymbol{S}(\boldsymbol{S}(0))))$

APROVE - syntax

"("

Declaration

")"

Program

VAR …

THEORY (Equations)

RULES …

STRATEGY …

Declarations

- Equations : *Term "==" Term.*
- RULES : *Term "->" Term.*
- Strategy : *INNERMOST*

  CONTEXTSENSITIVE (Variable int)

APROVE - syntax

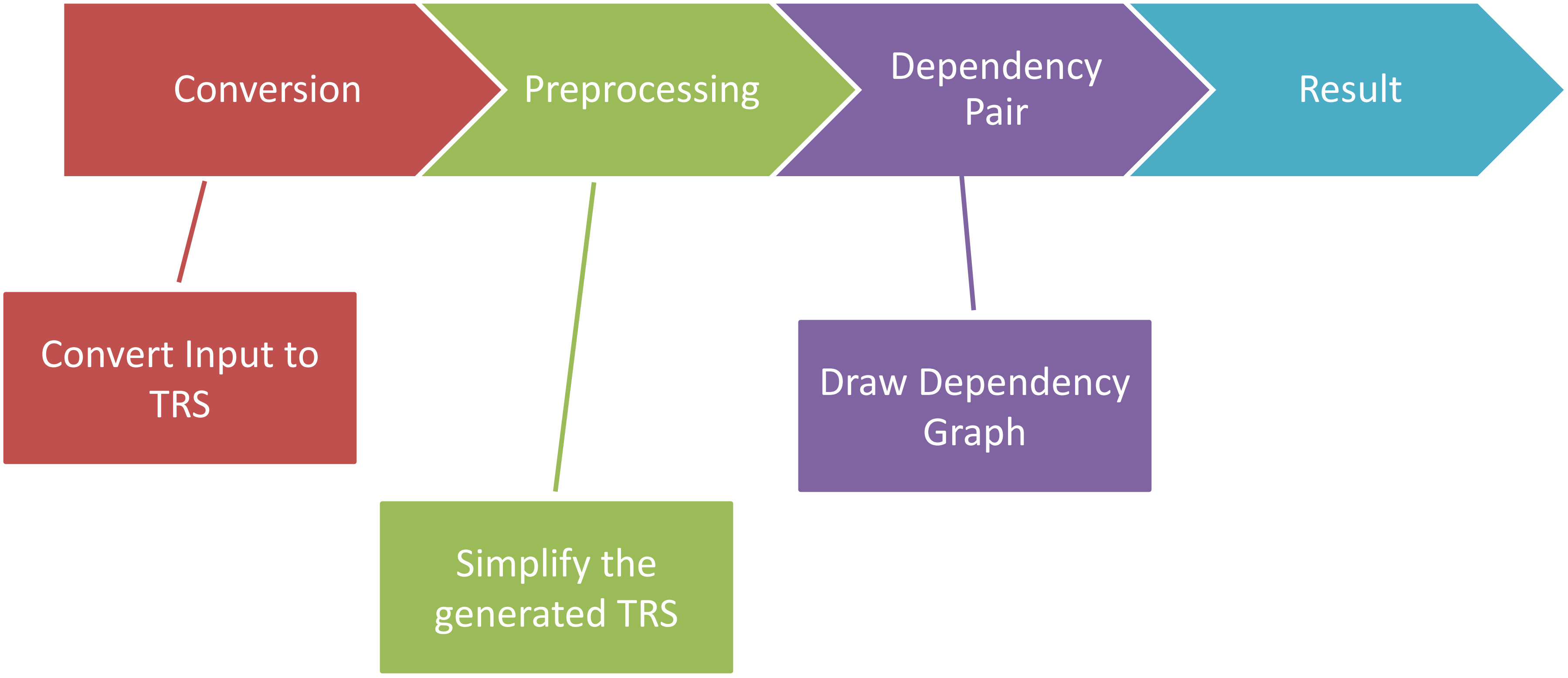
- Simple Example:

```
(VAR x y)
(RULES

  plus(s(s(x)),y) -> s(plus(x,s(y)))
  plus(x,s(s(y))) -> s(plus(s(x),y))
  plus(s(0),y) -> s(y)
  plus(0,y) -> y

  ack(0,y) -> s(y)
  ack(s(x),0) -> ack(x,s(0))
  ack(s(x),s(y)) -> ack(x,plus(y,ack(s(x),y)))
)
```

APROVE - proving mechanism



APROVE - preprocessing

- Aim?!
- Simplifying TRS for further processing

- Main techniques:
  - Removal of Redundant Rules (RRR)
  - Rule Reversal

APROVE - dependency pair

1. Construct List of Defined *“D”* symbols
   – Symbols present on the LHS

2. For every item in *“D”*, build list *“P”* containing all the rules *“A → B”* were *“B ∈ D”*

3. Repeat the same process on the output list

APROVE - dependency pair

- Example:

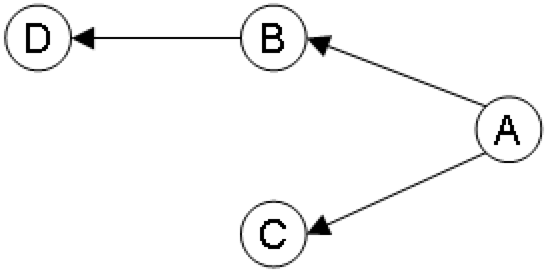
$$\mathrm{minus}(x, 0) \to x$$
$$\mathrm{minus}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{minus}(x, y)$$
$$\mathrm{quot}(0, \mathrm{s}(y)) \to 0$$
$$\mathrm{quot}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{s}(\mathrm{quot}(\mathrm{minus}(x, y), \mathrm{s}(y)))$$

- Dependency pairs:

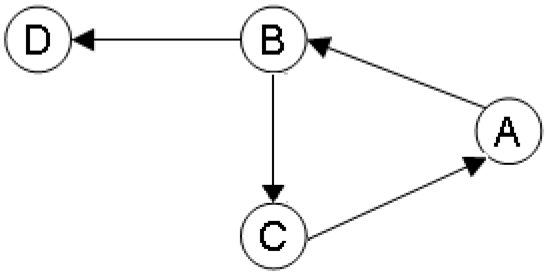
$$\mathrm{QUOT}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{QUOT}(\mathrm{minus}(x, y), \mathrm{s}(y))$$
$$\mathrm{QUOT}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{MINUS}(x, y)$$
$$\mathrm{MINUS}(\mathrm{s}(x), \mathrm{s}(y)) \to \mathrm{MINUS}(x, y)$$

APROVE - dependency graph

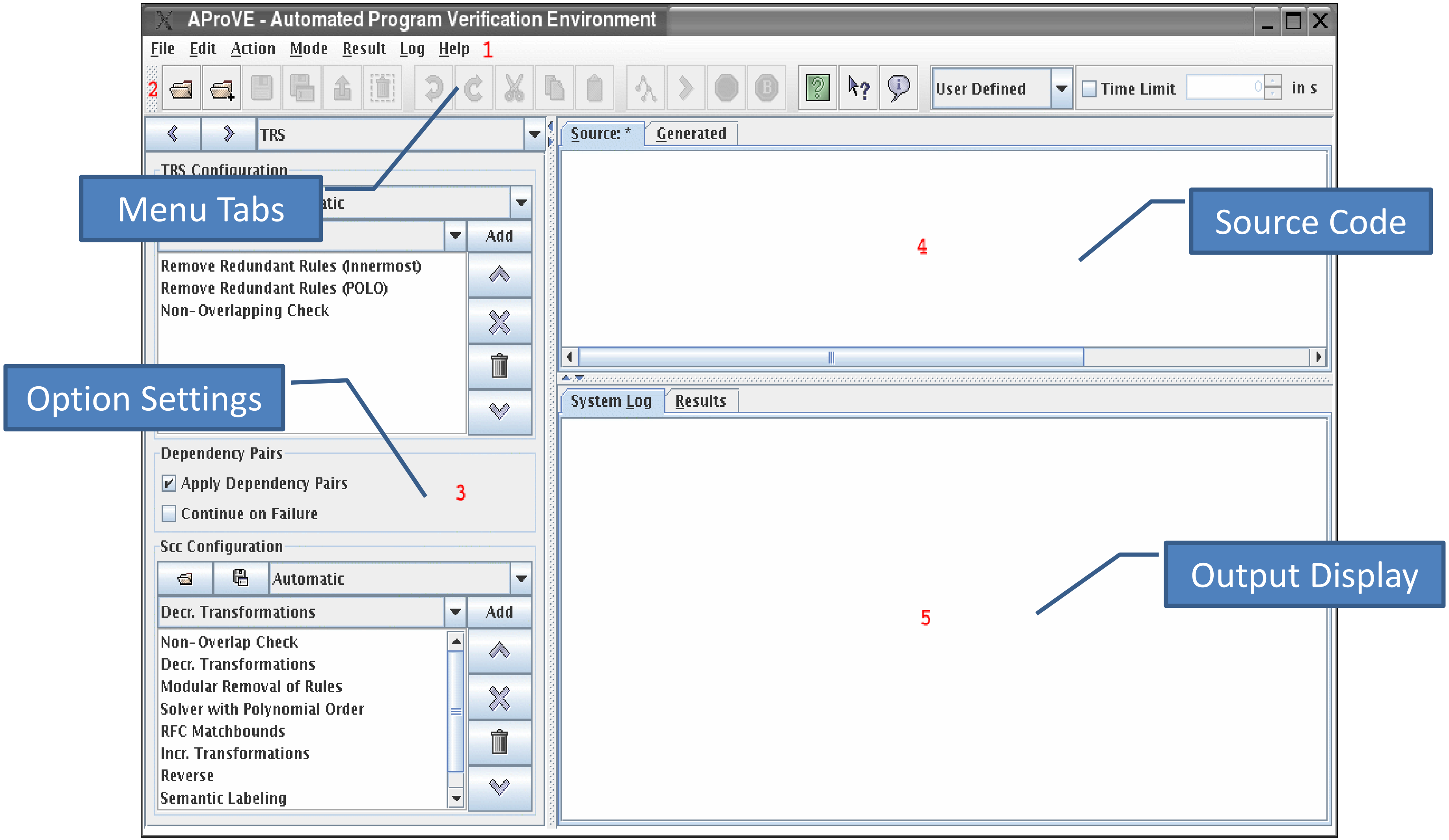
- (A,B) models a dependency "a needs b evaluated first“
  - Given: A = B+C; B =5+D; C=4; D=2
  - Then R = (A,B),(A,C),(B,D)



  - Given: A = B+5; B =C*D; C=4A; D=2
  - Then R = (A,B),(B,C),(B,D),(C,A)



APROVE - GUI



AProVE - Automated Program Verification Environment
File Edit Action Mode Result Log Help 1
2
User Defined
Time Limit
in s
TRS
TRS Configuration
Menu Tabs
Add
Remove Redundant Rules (Innermost)
Remove Redundant Rules (POLO)
Non-Overlapping Check
Option Settings
Dependency Pairs
Apply Dependency Pairs
Continue on Failure
3
Scc Configuration
Automatic
Decr. Transformations
Add
Non-Overlap Check
Decr. Transformations
Modular Removal of Rules
Solver with Polynomial Order
RFC Matchbounds
Incr. Transformations
Reverse
Semantic Labeling
Source: *
Generated
4
Source Code
System Log
Results
5
Output Display

APROVE - output

- The Output Consists of:-
  1) Proof Header:
     - *TRS Generated*
  2) Proof Body:
     - *Tree Like graph of techniques used*
  3) Proof Result:
     - *The Conclusion reached*

Questions?